https://theses.gla.ac.uk/

Theses Digitisation:

https://www.gla.ac.uk/myglasgow/research/enlighten/theses/digitisation/

This is a digitised version of the original print thesis.

Copyright and moral rights for this work are retained by the author

A copy can be downloaded for personal non-commercial research or study, without prior permission or charge

This work cannot be reproduced or quoted extensively from without first obtaining permission in writing from the author

The content must not be changed in any way or sold commercially in any format or medium without the formal permission of the author

When referring to this work, full bibliographic details including the author, title, awarding institution and date of the thesis must be given

Enlighten: Theses
https://theses.gla.ac.uk/
research-enlighten@glasgow.ac.uk

BATCH PROCESSING FOR TELETYPE ALGOL

by

WILLIAM SCOTT BOWIE

A Thesis submitted for the degree of Master of Science

at Glasgow University.

ProQuest Number: 10647839

All rights reserved

INFORMATION TO ALL USERS
The quality of this reproduction is dependent upon the quality of the copy submitted.

In the unlikely event that the author did not send a complete manuscript
and there are missing pages, these will be noted. Also, if material had to be removed,
a note will indicate the deletion.

ProQuest 10647839

Published by ProQuest LLC (2017). Copyright of the Dissertation is held by the Author.

All rights reserved.
This work is protected against unauthorized copying under Title 17, United States Code
Microform Edition © ProQuest LLC.

ProQuest LLC.
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106 – 1346

ACKNOWLEDGEMENTS

In submitting this thesis I should particularly like to thank Mr. G. I. Matkovits for his earlier work on the KDF9 Director subprograms and the Whetstone Compiler and for his constant guidance and encouragement. Thanks are also due to Professor Gillies for his advice and assistance and to all other members of staff both past and present who have contributed to this work in any way.

CONTENTS

CHAPTER 1

THE NEW SYSTEM

1.1 Introduction

Early in 1966, preparations were made for undergraduate courses in computing to start in October of that year. About 120 students were expected to enrole in the course and adequate facilities for the punching and running of student programs were necessary, together with an efficient operating system to allow each student to run his program at least twice in an afternoon.

Data preparation equipment was initially provided by the purchase of eight Westrex Model 33 Teletypes. In contrast to the situation at Manchester University, where a punching service is provided, (Brooker and Rohl, 1966), students punch their own programs. This decision was taken for two reasons; there were not enough operators to provide such a service and it was felt that students would benefit from the experience gained in doing their own data preparations.

To cope with the large number of programs produced, a batch processing system was implemented. Two programs were written to load programs and data onto magnetic tape for subsequent processing. The first was written as part of the KDF9 Non-Time-Sharing Director to perform the loading pseudo-off-line, thus making more use of the central processor. The second was written in the KDF9 assembly language, Usercode, to operate in a high priority level on the time-shared KDF9.

It is not possible to give a detailed account of the KDF9 in this thesis. A list of the relevant KDF9 manuals may be found in Appendix 4 under the heading 'KDF9 Documentation'.

1.2 Operating Systems

1.2.1 Introduction

The basis of an operating system is a control program (Supervisor, Executive) which calls into operation, and subsequently directs, programs which are required to be run on a computer. The rest of the system comprises 'system' programs such as languages compilers and sorting programs. In this section we shall examine operating systems from two points of view. Firstly, what contribution does the operating system make to the efficient running of the computer? Secondly, what kind of operating system is desirable for executing a large number of small jobs?

1.2.2 Background

Users of early computers did not have an operating system at their disposal. To use the computer, the programmer booked his time and had the machine entirely to himself during that time. If a program was being tested, the computer often stood idle while corrections were applied to the program before it was run again. This type of system existed until a few years ago on computers such as the EDSAC and DEUCE, and is still used on small machines like the SIRIUS.

One of the problems which confronted users of such machines was the difference in speed between the peripheral devices and the central processor. With the advent of faster transmission rates and autonomous, asynchronous operations - allowing simultaneous input/output and computing - the user's problem appeared to be solved. The increase in speed, however, only

served to emphasize the amount of time wasted by the user in changing magnetic tapes, logging on and off and so on. More sophisticated input/output also made efficient programming more difficult. To utilize the new hardware facilities, Input/Output Control Systems (IOCS) were developed. The IOCS relieved the programmer of the complexity of efficient input/output programming. To exploit the features of the IOCS, programs were assembled in batches and users were replaced by full-time operators to organize the assembly and running of batches. The simplest form of batching was to stack a number of jobs punched on cards in the card reader. By assembling the jobs on magnetic tape, a faster throughput could be achieved. This method, however, has the disadvantage of requiring an extra run to assemble the batch. To schedule and sequence the running of batches control programs were developed. These incorporated an IOCS and assembling batches on magnetic tape could be performed pseudo-off-line through the supervisor program.

1.2.3 Current trends and problems

The improvements in computers and operating systems still leave unsolved the problem of loss of time between jobs. The central processor is much faster than any peripheral device and it can be idle while a program is being loaded into the machine. If a program is expected to run for 20 minutes then a loading time of 10 seconds is not important. However, if the loading is 5 seconds and the program computes for 5 seconds, the loading time becomes an important factor in the efficient

utilization of the computer, particularly if there are a large number of such small jobs.

As computers increase in size and speed, yesterday's large job becomes today's small job, and yesterday's small job becomes tiny. The problem facing many installations is the wide range of jobs which must be catered for. Jobs may range from a five hour program which requires four magnetic tapes to a ten second program which uses only a paper tape reader and a paper tape punch. To provide for such diversity, operating systems must be sophisticated and more complex.

Time-sharing (or multi-programming) adds even more to the complexity of the operating system. By having several programs in core at any one time, control may pass from one program to another whenever a program is held up. In this way the central processor can be kept busy for most of the time, but the presence of several programs in core store at the same time raises further problems. Allocation of core store and peripherals, deciding on priorities and protection of programs must all be taken into account. The penalty one pays for a more efficient system is a more complex supervisor program to cope with these problems.

Even with a more complex system, the wide range of jobs must still be catered for and this is complicated by the fact that several different jobs will be in the machine at the same time. The size of a program, its peripheral requirements, the time it will take and its priority, all contribute to the difficult task of scheduling. With a scheduling algorithm, this work could be done by the supervisor program, adding even

more to the already complicated system. If the supervisor is to do the scheduling, it must be supplied with a list of jobs from which it can choose those which can be run at any particular time.

The Manchester University Atlas computer has an 'Input Well' system. (Kilburn et al., 1961). An area of core store is filled with jobs by slow peripherals operating in parallel, and by magnetic tapes. When a level of time-sharing becomes available a job is selected from a list kept by the supervisor program. The use of such a well tends to level out the rate at which information is required by the central processor. Results are sent to a similar 'Output Well' for distribution to the output peripherals. In this way the central processor and many of the peripherals are kept busy.

1.2.4 The system and the user

So far we have been concerned with the operating system as it affects the running of the computer. The operating system is designed to make full use of the hardware facilities and to avoid wasting machine time. On the other hand, how does the operating system affect the user's turn-round time? One would hope that the system would reduce this time but this is often not the case. As operating systems become more complex, more and more time is spent in organising the flow of jobs through the machine, and a program will thus take longer to run on a time-sharing machine than it would have on a non-time-sharing computer. If the supervisor program selects the jobs to be done, the turn-round time could be much longer than the programmer anticipated. For example, in the Input Well system,

there may be a number of small jobs all of which require the same compiler. These jobs may have to wait some time before they are attended to, since if there is a large program occupying the machine, there may not be room for the compiler. Whereas a user with a five hour job may not mind waiting 24 hours for his results, a student who submits his 10 second program at 10 a.m. will certainly not wish to wait until 5 p.m. for his results.

1.2.5 Subsystems

It would appear that no matter how sophisticated the computer and its operating system, there is a case for having different types of operating systems for different types of jobs. Since such operating systems would in fact be part of the general system, a better term might be 'operating subsystem'. As we have seen, it is the small jobs which are not really handled very well by current operating systems. On a non-time-sharing machine, time is lost because the loading time is a large part of the total run time. Time can also be lost on a multi-programmed computer because the job cannot be run until there is enough room in the machine for the required compiler.

Such small jobs are better run in a batch. With a batch processing system on a non-time-sharing machine there are two immediate advantages:

(1) The programs are loaded onto magnetic tape pseudo-off-line thus making more use of the central processor.

(2) Processing the batch from magnetic tape will decrease the loading time.

On a time-sharing machine, the programs may be loaded by a program running

in a high priority level, and the batch run in the same level. In this way, the loading time (including the time taken to bring in compilers) is absorbed by lower priority programs.

Such a system, which forms an operating subsystem, has been implemented on the KDF9. Before discussing batch processing on the KDF9, it will be necessary to consider the KDF9 and its operating system in the light of the above discussion.

1.3 KDF9 Computer

Although a complete description of the KDF9 cannot be given here, it will be advantageous to consider some of the more important features of the hardware and logic of the computer.

1.3.1 Main store and instructions

The main store of the KDF9 is arranged in modules of 4096 words, each word containing 48 bits. A maximum of eight modules may be fitted, giving a maximum core store of 32,768 words. A word may be thought of as six 8-bit syllables and machine code instructions may occupy one, two or three syllables.

1.3.2 The Nesting Store

Arithmetic operations are carried out in the 'nesting store' which is a push down stack of 16 words of core store. Physically, the nest consists of three special registers N1, N2 and N3 in addition to the stack, a total of 19 words, or 'cells'. When the KDF9 control program (Director)

is entered, N1, N2 and N3 are pushed down into the stack and other programs are thus restricted to using only 16 cells.

1.3.3 The Subroutine Jump Nesting Store

This nesting store, usually referred to as the SJNS, is used to store the return address every time a subroutine is entered. It is similar to the arithmetic nesting store but only has one special register in addition to the stack of 16 cells. Programs may use up to 14 cells of the SJNS, two being required when Director is entered, (one is used for the program return address). Communication exists between the top cell of the SJNS, called SJNS1, and N1 to allow ~~address~~ addresses to be inserted or erased. A subroutine return address, or strictly speaking, the contents of SJNS1, is referred to as the 'LINK'.

1.3.4 Q Stores

Attached to the arithmetic nest is a set of 16 special registers known as Q stores. These may be used for a variety of purposes such as preserving data or results when the nest might otherwise become overfilled and as index registers for modifying main store addresses. A Q store can be regarded as a 48-bit register or as consisting of three 16-bit registers. These three sections are known as the Counter (C), Increment (I) and Modifier(M). Extensive use is made of this breakdown of the Q store particularly in input/output operations. Q stores are numbered Q0 to Q15. The hardware ensures that Q0 always contains zero.

1.3.5 Input/Output

Information can be transferred to and from the machine in a variety of ways. Paper tape, cards and magnetic tape can be used for both input and output and results can be printed on high speed printers. To perform any input or output transfer on KDF9 it is necessary to specify a device to be used and a quantity of data to be transferred. This information is provided by the programmer in a Q store which is most commonly laid out as follows:

Counter : The device number.

Increment : Lowest main store address of the area concerned with the transfer.

Modifier : Highest main store address of the area concerned with the transfer.

Peripheral devices have two numbers associated with them. One, known as the 'type' number, specifies the kind of device (e.g. paper tape reader) and the other is the actual 'device' number or 'unit' number. Type numbers are octal integers in the range 00 - 77 and the type number of a device can be changed by the computer operator.

This feature allows great flexibility in the use of input/output devices. For example, if the line printer is unavailable, the type number of the paper tape punch can be changed to that of the line printer and results output to the punch as though it were a printer.

1.3.6 KDF9 Director

The control program of the KDF9 is known as 'Director'. It occupies

approximately 2,000 words of core store starting at absolute address 0. The various functions which Director performs include supervising the running of programs by allocating devices for use by the program, recording 'real' time and program run time, and handling various input/output transfers. The KDF9 Non-Time-Sharing Director is discussed in more detail in section 4.

1.3.7 Monitor Typewriter

Control of the machine is exercised by the operator by means of an on-line Flexowriter known as the monitor typewriter which operates at 10 characters per second. All messages concerned with the running of programs, and requests by Director for specific operator action are output to the monitor typewriter. The operator communicates with Director by pressing an 'interrupt' key on the typewriter.

When reference is made to any characters which are typed on the monitor typewriter, these will be enclosed by the brackets < and > .

1.3.8 Time-Shared KDF9

Time-sharing facilities are available for the KDF9 enabling up to four programs to be stored in core at any one time. Control passes from one program to another whenever a program is held up, according to a priority number assigned to each program. The configuration of the Time-Shared KDF9 and its operation is discussed in Chapter 3.

1.3.9 Programming Languages

Usercode is the alphanumeric assembly language of the KDF9. The subprogram DG was written in Usercode and tested in program mode before being inserted into Director.

Paper Tape Generator Code (PTG) is similar to Usercode but provides some extra facilities. The KDF9 Non-Time-Sharing Director is written in PTG and the subprogram was re-coded in PTG after being tested in Usercode. Descriptions of Director and the subprogram DG necessarily refer to labels and constants and an explanation of the terminology is required.

In PTG, reference labels and symbolic addresses consist of up to four alphanumeric characters between oblique strokes e.g. /ABCD/ , /176/. For the purposes of description, the obliques will be omitted when reference is made to constants and buffer areas, so that, e.g. BUF1 is a buffer area but /ABCD/ is a reference label (or the start of a subroutine).

1.4 KDF9 Non-Time-Sharing Director

1.4.1 Introduction

The KDF9 computer operates in two modes - in 'Program mode' and in 'Director mode'. The machine is set in Director mode by either

(1) Initial input of Director at the start of a day's run, or

(2) An 'interrupt' while the machine is in Program mode.

1.4.2 Interrupts

There are two types of interrupt into Director - a programmed interrupt

by means of the Usercode instruction 'OUT' which is a request for Director to take some action on behalf of the program (e.g. allocating a peripheral device) - and hardware interrupts such as the automatic clock interrupt every 1.04 seconds and the typewriter interrupt (TINT) caused by the operator pressing the interrupt button on the monitor typewriter.

When an interrupt takes place, Director examines the 'Reason for Interruption' Register (RFIR) and takes the appropriate action, returning control to the program only when all interrupts have been serviced, since other interrupts may occur while the first is being attended to.

1.4.3 Director's Use of Stores

In the standard non-time-sharing KDF9, there is only one set of nesting stores, SJNS, Q stores, one Test Register and one Overflow Register available for use by both program and Director. Should any of these be required by Director, their contents must be preserved in core store locations and restored before control is returned to the program.

However, due to the physical arrangement of the nesting stores, Director may use up to three cells of the nesting store and one cell of the SJNS without endangering the program.

1.4.4 Input/Output in Director Mode

Each module of core store is laid out in blocks of 32 words, and whenever an input/output operation takes place, all blocks of 32 words

either partly or wholly engaged in the transfer become 'locked out' and the buffer associated with the peripheral device becomes 'busy'. In program mode, if any instruction (except those available for testing for these conditions) refers to a busy peripheral device or a locked out area of store, an interruption known as 'Lock-Out Violation' (LOV) occurs and the execution of the instruction is postponed until the busy/lockout conditions are cleared.

However, in Director mode the situation is different, since no LOV interruption takes place. Thus if Director initiates a transfer and subsequently finds itself in a situation which, in program mode, would lead to a LOV interruption, it must suspend the current activity and do something else until the busy/lockout conditions are cleared, when it can resume the suspended activity. If there are several of these activities which can be suspended and resumed, a 'programmed time-sharing' system must be organised within Director.

The main hardware aid to such a system is the 'End Director Transfer' (EDT) interrupt. Since the suspension of an activity is almost always due to a Director-initiated transfer, the EDT interrupt signals the potential end of the suspension and it is then the task of the system to discover which activity was held up and where to resume it.

The routines involved in this 'programmed time-sharing' system are known as 'Subprograms'.

1.4.5 Subprograms

Subprograms are subroutines within Director concerned with various

activities. Whenever one of these subprograms requires access to a busy peripheral device or a locked out area of core store the system which controls the subprograms is capable of suspending the routine until the busy or lock-out conditions are cleared. The basis of the system is the 'subprogram parameter table'. This consists of three words for each subprogram indicating where the subprogram is held up, and why. This subprogram parameter table is addressed by Q7.

When an EXT interrupt occurs, the subprogram parameter table is examined to see if the first subprogram can be resumed. If it cannot be resumed, the next entry is examined to see if the next subprogram can be resumed, and so on. If all subprograms are either held up or have finished, control returns to /EW9/, that part of Director which examines the EXIN to see if any more interrupts have occurred, before returning control to the main program.

Once a subprogram has been resumed, it proceeds until one of several things happen:

(i) The subprogram finishes and control passes to the next subprogram.

(ii) It fails and control passes to /EW9/.

(iii) It suspends itself until the next EXT interrupt.

(iv) It suspends itself if a peripheral device is busy and/or an area of core store is locked out, by jumping to routine /EW15/.

The system also caters for a hold-up due to a request for a magnetic tape which is not immediately available.

1.4.6 Extension of the Subprogram System

In the summer of 1965, the subprogram parameter table was extended and parts of Director modified to allow new subprograms to be inserted into Director.

A new subprogram, written in Paper Tape Generator code is compiled with Director into the area between reference labels /A000/ and /A111/. By means of the instruction 'BLANK', blank tape is generated at the beginning and end of the binary version of the subprogram and in this way it can easily be distinguished from the rest of Director and further copies of it made.

Once the new Director has been read into the machine, the subprogram can be read in to its predefined area of store by means of a typewriter interrupt. When the interrupt button is pressed and the query <TIWT;> appears on the monitor typewriter, the operator types <DI.→> . A paper tape reader (type number 67) is allocated for Director's use and the subprogram is read in from this paper tape reader. When the transfer is finished, the subprogram can be started by a similar typewriter interrupt, the operator replying <DE.→> to the <TIWT;> query.

This method allows different subprograms which carry out different tasks to be compiled as part of Director. Each subprogram must be compiled with the complete Director but it is not necessary to load a new Director in order to use the new subprogram, as long as it does not occupy a greater area of core store than the subprogram originally compiled with the Director currently in control of the computer.

1.5 Whetstone Algol

1.5.1 Introduction

A large number of the jobs run on the KDF9 are short Algol programs. These programs are run on a load-and-go basis, using the Whetstone Algol Compiler (Randell and Russell, 1964; Wegner, 1964). This compiler consists of two parts - a Translator and a Controller. Algol programs are translated into an intermediate object code which is then obeyed interpretively by the Controller. The translation is extremely fast but the object program produced is slow to run. The system is therefore used to develop and debug programs.

Short paper tape Whetstone programs, i.e. those which run for less than three minutes, may be run twice each day - in the morning and the afternoon. With this system, which is a batch processing run since the Translator is re-entered from the Controller, a typical batch of 20 programs takes about an hour to run. Results are stored on magnetic tape for later pseudo-off-line printing. The length of time a programmer has to wait for his results naturally depends on the size of the batch but the average turn-round time is two hours.

1.5.2 Batch processing

Initial steps to reduce this turn-round time and make the use of the KDF9 more efficient were taken in 1965 when the subprogram facilities in Director were extended. A subprogram was written to load Whetstone Algol programs and data pseudo-off-line onto magnetic tape. The Whetstone compiler was modified to read from magnetic tape instead of paper tape.

With the programs on magnetic tape, a batch could be processed in a much shorter time and the programmer's turn-round time therefore decreased.

Although the system proved satisfactory, it was not often used, mainly because of trouble with the paper tape reader used for the off-line loading. On other occasions there were not enough short programs to make the batch processing run worthwhile.

1.5.3 Student programs

At the practical classes in the undergraduate computing courses, students prepared and punched Algol programs. A rapid turn-round time is necessary to allow students to debug and develop their programs. Programs and data were punched on Westrex Teletypes. However, to run these programs under the normal Whetstone Algol system would have involved modifications to the compiler to accept Teletype code and at the same time would not have provided the rapid turn-round time desired. A batch processing system based on the earlier work was necessary to cope with the large number of student programs and make more efficient use of the central processor by utilising the extended facilities available in Director.

A new subprogram, known as DS, was written to load Teletype paper tapes to magnetic tape for subsequent processing by the modified Whetstone Compiler. During the loading stage, a conversion from Teletype code to KDF9 paper tape code takes place so that the programs on magnetic tape are in the required code and format for the compiler.

CHAPTER 2

TELETYPE ALGOL

2.1 Paper Tape Codes

2.1.1 KDF9 Flexowriter Code

Flexowriter Code (or simply KDF9 code) is 8 channel, even parity, the parity bit being in channel 5. Characters are formed from six bits, giving 64 distinct symbols. Altogether, 128 characters are provided, although some of these are non-printing. This is achieved by two special characters, 'Case Shift' and 'Case Normal'. The 64 characters able to be formed by six bits are interpreted in one of two ways depending on which 'case' character precedes them. We thus have 'case shift characters' and 'case normal characters'. Certain characters such as 'space' have the same meaning irrespective of the case in which they appear. Channel 8 is only punched for the 'space' and 'erase' characters.

2.1.2 Teletype Code

This code is also 8-track even parity, with parity in channel 8. There are thus seven information bits, allowing 128 characters, but not all of these are available on the Model 33 Teletypes. Two special keys 'Control' and 'Shift' are used to obtain different characters; pressing the 'Shift' key inhibits punching in channel 5 while preserving parity by inserting or removing channel 8 as required. The 'Control' key removes channel 7 and similarly preserves parity. Certain keys are locked out when control is pressed. Control characters are not used since they may be confused with other characters, e.g. control K has the same pattern as

'Return'.

Full details of these codes may be found in Appendix 1.

2.2 Representation of Algol on Teletypes

Strictly speaking, we are not only considering the representation of Algol, but also the representation of the KDF9 paper tape code on Teletypes since a translation from the latter code to Teletype code is carried out. The problems of representation are discussed from an Algol viewpoint, the batch processing system having been designed for Wheatstone Algol programs.

The representation of a programming language within a given character set involves two considerations:

(i) The character set should be capable of representing all the basic symbols of the language.

(ii) This representation should be clear to the user, i.e. the meaning or function of a particular symbol as typed should be obvious to anyone reading the program.

These considerations are closely related. The first can usually be satisfied if those symbols which do not appear in the character set can be represented by other characters or strings of characters. The second consideration requires that this type of representation should be clear, in that the characters should bear some relation to their functions.

Consideration (ii) may appear in some respects to be irrelevant, since it would not seem to matter which characters are used in a representation as long as a list of these symbols and their functions was provided for the user. On the other hand, the batch processing system is provided as an aid

to undergraduate teaching, and if the appearance of a symbol when typed bears some relation to its function, the teaching of Algol will be simplified.

Before proceeding to Teletype representation, we shall illustrate these considerations with examples from the Creed 5 channel version of KDF9 Algol.

In 5 hole tape code there is no obvious character for 'to the power' and it is represented by **. This satisfies condition (1) but the meaning of this compound character is not obvious. Similarly, *> is used to mean $\leq$ and again the appearance of the compound character does not really bear any relation to its function.

However, the 8 channel Teletype code affords more characters than 5 channel and the problem of choosing a suitable version of Algol for Teletype keyboards is thus simplified. Even so, there are certain Flexowriter characters which do not appear on Teletypes. These may be more clearly represented by two or more Teletype characters, rather than by a single character whose meaning is not obvious. An example of this is the Teletype version of integer divide ($\div$), which is typed as "/" rather than $\frac{1}{8}$ or $\frac{1}{2}$.

To consider Teletype representation in more detail, various basic symbols are grouped according to their function. Characters common to both Teletypes and Flexowriter are capital letters, numerals and the characters £ / * - ^ : ; . , subscript 10. Any symbols not specifically mentioned may be assumed to have the same appearance on both keyboards.

2.2.1 Alphabetic Characters

Flexowriter representation of Algol allows the user to type both capital letters (case normal) and small letters (case shift). Teletypes have only capitals and the convention adopted is that these capitals will represent small Flexowriter letters. Algol programs consist predominantly of small letters; these are in fact required for standard procedures such as 'read', 'sin' etc. Certain strings of letters must, however, be in capitals; program identifiers and magnetic tape identifiers come into this category. Such strings are preceded by %, e.g. %ROOLSOOYABED for a program identifier. The occurrence of numbers in such strings is permitted since their Flexowriter equivalents are typed in case normal anyway. For the purposes of code conversion these strings are referred to as 'special strings'.

2.2.2 Underlined words

The term 'underlined words' is used here to describe those Algol delimeters formed in Flexowriter Algol by underlining a string of letters. Some examples are begin, real and step. There is no facility on Teletypes for underlining and some other method of representing these basic symbols is required. The string is placed within double quotes as in "BEGIN". In two or more consecutive underlined words, the inner quotes may be omitted, as in "BEGIN REAL PROCEDURE".

2.2.3 Arithmetic and Relational Operators

There are five operators which require special representation on

Teletypes; three relational operators and two arithmetic operators. The arithmetic ones are multiply and integer divide (÷). The latter we have already seen is typed as "/", i.e. real division bracketed by quotes. For multiply, we may use *, a symbol fairly commonly used for this purpose. KDF9 code, however, uses * for space in strings and a replacement for this character becomes necessary if we use * for multiply.

The relational operators $\leq$ and $\geq$ are produced on the Flexowriter by underlining the inequality. Since underlining is not available we may follow the convention adopted for underlined words and type these inequalities as "⊏" and "⊐". In the interests of clarity, however, it was decided to allow users to type these symbols as <= and >=.

The Flexowriter character 'not equals' (≠) follows a similar convention whereby the user types != on Teletypes.

Spaces may occur in these compound symbols, as in ! = or " / ".

2.2.4 Other Characters

String quotes, ⊑ and ⊒ on the Flexowriter, are typed as ` and ′. An acceptable alternative representation would be "⊏" and "⊐", but the former is preferred because of its similarity to the Algol 60 reference language (although the accents are the opposite way round).

A Teletype character corresponding to * for space in strings is required if * is used for 'multiply'. The dollar sign ($) was selected since the 'S' may be associated with the idea of a space.

To obtain a new line, the user types 'Return' followed by 'Line Feed'. The KDF9 End Message (→) is typed as a backwards arrow (←).

The ampersand (&) is used as a special warning character to denote the beginning of a program and the character ½ is ignored if detected during code conversion.

2.2.5 Context Sensitivity

It will be noted that the representations outlined above are not context-free, in that quotes (") is used to mean different things. At the beginning of a word the symbol means 'start underlining' and at the end of a word, 'stop underlining'. The Teletype version of ÷ also involves this character. If possible, a representation should not contain anomalies of this kind, but in choosing the Teletype version of Algol, considerations (i) and (ii) at the beginning of this section were given high priority, since it is possible to examine the context in which a character appears during the conversion to Flexowriter code.

2.2.6 Summary of Representations

The table on the next page shows those symbols and characters which required special representation on Westrex Teletypes. A specimen Algol program printed in both KDF9 Flexowriter representation and in Teletype representation can be found in Appendix 2.

Flexowriter	Teletype	Comments
begin	"BEGIN"	underlined words.
⌊	\	open string quote.
⌋	/	close string quote.
≤	<=	less than or equal to.
≥	>=	greater than or equal to.
≠	!=	not equal to.
×	*	multiplication
÷	"/"	integer division.
#	#	space in strings.
→	←	end message
CRLF	Return, Line Feed	newline.

2.3 Problems of Implementation

2.3.1 Introduction

In this section, we shall consider some of the problems encountered in implementing the batch processing system. The subprogram DB, operating in Director mode, naturally must conform to the general rules for Director mode programs. The representation of Algol, too, creates problems of input and of context analysis. The areas where most consideration was required were Director and paper tape input.

2.3.2 Problems associated with Director

Any program written to operate in Director mode is subject to the general rules laid down for Director. As mentioned in section 1.4.3, it is possible for Director to use 3 cells of the nest and 1 cell of the SJNS without endangering the program. This means that if more than 3 cells of the nest or more than 1 cell of the SJNS are required some cells of those nests must be dumped into core locations. This is done frequently in DB in order to test the value of a character or when calling a subroutine from within a subroutine.

Similarly, before an input or output transfer is initiated, it is necessary to 'clear' the nests, i.e. any cells of the nests being used by the subprogram must be preserved. Once the transfer starts, the subprogram is suspended, and the machine will revert to program mode if there are no other active subprograms and no more interrupts to be serviced.

When an interrupt (other than LOV) occurs, Q stores 5,6,7 are preserved by Director for its own use. Any other Q stores required by subprograms must be preserved locally and the contents restored before initiating any peripheral transfers. The use of Q stores, therefore, is subject to the same conditions as the use of the nesting stores.

The clock interrupt, once every 1.04 seconds, is used by Director to record real time and program run time. If a second clock interrupt occurs before Director clears the first, a 'Reset' interrupt causes Director itself to be interrupted and a catastrophic machine failure results. From the programming point of view, this means that loops of instructions must not take more than one second to be executed. Care must therefore be taken to ensure that the subprogram does not get held up in any loops.

2.3.3 Paper Tape Input

When paper tape in KDF9 code is read into the computer using the standard reading instructions (PRQq, PRBQq), channels 5 and 8 are removed by means of a plug attached to the paper tape reader and the resulting six-bit characters are packed eight to a word. Reading Teletype code (which has parity in channel 8 and information in channel 5) on this basis, however, results in ambiguity in the internal representation of characters, as the following example shows:

'space has the bit pattern 10100.000

and 'zero' has the bit pattern 00110.000, where '.' indicates the position of the sprocket holes. Removing channels 5 and 8 leaves the six-bit pattern

01000 which can represent either character.

One method of removing this ambiguity is to read all eight bits of a character. The instructions provided for this purpose (PRONq, PRONq) read one eight bit character into one main store word. Thus to read, say, 256 characters in one input operation, we would require a buffer of 256 words, which due to the limited space available (approximately 500 words) in Director for the subprogram, is too large. In addition, the use of eight bits involves a larger table for code conversion.

Using input instructions of the form PRNq to read paper tape punched on Teletypes will result in ambiguity in the internal representations no matter which channels are removed, since the tape code has seven information bits. However, the character set on the Model 33 lies in the middle of the full set and channel 6 can, in fact, be regarded as 'space'. There are two characters which lie outside the main body of the table; these are 'Return' and 'Line Feed'. Removing channel 6 (effectively subtracting 32 from the value of the character) means that the characters 'minus' and 'multiply' (*) have the same value as Return and Line Feed respectively. Since Return is usually associated with Line Feed, it is not too difficult to recognise these characters if we stipulate that they must always appear together. Thus, to obtain a new line, the user must type Return followed immediately by at least one Line Feed. Any number of Line Feeds may in fact be typed after Return, but on converting to EBCDIC code, only one CRLF character is produced. Should the user actually type minus, multiply (-*), which in any case is invalid Algol, a CRLF character will be produced.

This rule, however, imposes a restriction in that multiply may not be the first character of a line since it will be interpreted as another Line Feed.

Thus, when reading tapes produced on Teletypes into the KDF9, channels 6 and 8 are removed. This involves using a special plug, with channels 5 and 6 interchanged.

Standard input routines for the KDF9 use the instruction PRBq which functions in the same way as PRq except that the transfer stops when either the buffer area is filled (as for PRq) or when an 'End Message' character is read. The End Message character (→, octal value 75 in KDF9 code) appears at the end of a program identifier, at the end of a program and at the end of a data tape. Removing channels 6 and 8 from Teletype paper tape code results in the right square bracket (]) having an octal value of 75. Using the instruction PRBq, therefore, means that every input transfer would stop whenever the bracket was read. These brackets occur frequently in Algol programs, in arrays, and reading paper tape on this basis would be inefficient, to say nothing of the effect on the paper tape readers having to stop and start every few characters.

Having eliminated the instructions PRq, PRCBq because of the large buffer areas required to perform an efficient transfer, and the instruction PREQq because of the effects just discussed, leaves us with the instruction PRBq, which requires that the buffer area be filled before the transfer stops. During the loading of programs, in order to separate one program from the next, the buffer can be filled with End Message characters.

This method was employed in an earlier version of the system but was discarded because of its obvious inefficiencies. A more satisfactory and desirable approach would be to read programs and their data in succession. The problem, however, arises of separating programs within the buffer. How do we know where one program stops and another one begins? The End Message character does not help, since it can appear in three positions in the program and its data. In addition, forgetting to punch the End Message is one of the most common errors made by students (and others!). The requirement is for a special character to denote the beginning of a program, in order that the markers and counters used by the subprogram can be reset to their original states and that the new program cannot be affected by the one just written to the magnetic tape.

The character selected for this purpose is the ampersand. Each program must begin with two ampersands; this allows for the possibility that a single ampersand appears by error somewhere else in the program. This character is also used to terminate the off-line loading. When all programs and their data have been written to magnetic tape, an 'End Run' tape containing four ampersands and a large number of End Messages is read to fill the buffer at the end of the batch.

2.3.4 Magnetic Tape

Programs and data must be written to magnetic tape in the format required by the Whetstone Compiler for correct translation and execution. Program text is written in blocks of 32 words and data in blocks of 30

words, with a one word data separator block. The Wheatstone Controller which reads the data from magnetic tape was originally designed to cater for both 5-hole and 6-hole data. For 6-hole data, the data block is followed by a word of all zeros and for 5-hole data, a word of all ones.

To separate programs on magnetic tape, a one word separator block is written at the end of each program and three of these blocks (two if End of Tape Warning is detected) are written to the tape at the end of the run.

2.3.5 Output of results

Results of programs run in the batch processing system were printed on the Department's Holley Line Printer which differs from the standard KDF9 printer in having more characters on the barrel. Strings of text sent to the printer are usually strings of case normal characters. If the character Case Shift occurs, it is replaced by a dummy character. Teletype letters are converted into Flexowriter case shift (i.e. small) letters. Unfortunately, the dummy character inserted in place of Case shift prints as an 'X' on the Holley printer. This gave rise to confusing results. For example the statement

WRITE TEXT (70, \W.% S.% BOWIE./); printed as XW. XS. XBOWIE.

Users were advised to precede any text output by a % so that the resulting Flexowriter characters would be in case normal. We would thus rewrite the above statement as

WRITE TEXT (70, \%W.% S.% BOWIE./); to obtain W. S. BOWIE.

The 'space in strings' character (ϕ) has its KDF9 code equivalent (*) in case shift and special action had to be taken during code conversion when this character was recognised, since it could occur within a string of case normal characters.

2.3.6 Special Character Routines

These routines are concerned with the context analysis mentioned in the introduction. Before an equivalent Flexowriter character can be produced, it is necessary to examine subsequent characters when a special Teletype character is found. Examples of these characters are Return (which might be a minus if it is not followed by Line Feed), space in strings' (ϕ) and quotes ("). These special characters are discussed in the description of the subprogram D8 in the next section.

2.4 Director Subprogram D8

The subprogram is written in Paper Tape Generator code and is compiled with the standard non-time-sharing Director and loaded into the machine as described in section 1.4.6.

The subprogram consists of six basic sections as follows:

(1) Initial setting up of buffers and devices.

(2) A 'Look-at' table for converting Teletype characters to Flexowriter characters.

(3) Routines dealing with different characters.

(4) Input/output routines and preserving of Q stores.

(5) Termination steps.

(6) Constants and buffer areas.

2.4.1 Initial setting up

This part of the subprogram claims a paper tape reader (type 67) and a magnetic tape (type 60) using the Director subroutine /RAU/. If a paper tape reader of type 67 is not available control passes to /RESP/ and the subprogram is closed down. If the magnetic tape type 60 device is not available, the termination steps are entered to deallocate the reader before control passes to /RESP/. A type 67 paper tape reader is included in Director's peripheral device list and is normally always available. However, if a type 60 magnetic tape is not available, due perhaps to the operator omitting to change the type number, the subprogram can be restarted when the type number has been changed, by another <TINT; D2.→>.

Having claimed the input and output devices, the markers DATA, MTBC and EMMA (see 2.4.6) are set to zero and the first block of 32 words read in from paper tape. The contents of Q3 and Q4 are preserved by /QSD/ and the input and output buffers set up for the fetching and storing of characters.

2.4.2 Table Look-at

The table used to convert characters in Teletype code to Flexowriter code begins at reference /100/; in the subprogram and contains 64 one-word

entries. Each word contains either one or two 3-syllable instructions. For warning characters (such as &) and characters which have no Flexowriter equivalents (such as ½) the entry in the table consists of a jump instruction which transfers control to the routine which deals with the character. For other characters there is a 'SET' instruction followed by a jump instruction. The SET instruction places the value of the Flexowriter character in 81 before the jump is obeyed. The following section of the table should illustrate these points.

J/70/; *SET22; J/9/; SET30; J/6/;
SET31; J/6/; SET26; J/6/; SET35; J/7/;

Since the jump instruction only occupies 3 syllables of one word, an asterisk is placed in front of the next instruction. When the program is being compiled, this has the effect of filling out the rest of the word containing the jump with dummy instructions and ensures that the next entry starts at the beginning of the next word.

The table is accessed by the instruction EXIT , using the value of the Teletype character. The Teletype character is placed in the top cell of the SJNS and when the instruction EXIT/100/; is obeyed, the contents of SJN31 are added to the address of reference /100/ and control passes to the appropriate entry in the table. For example, 'space' has the value zero and leads to the first entry while " has the value 2 and leads to the third entry. The sequence of instructions to access the table is at reference /2/.

2.4.3 Routines dealing with different characters

Several of the Teletype characters used in the representation of Algol have special significance, or are part of multi-character symbols. It is therefore necessary to take specific action when these characters are detected, and routines to deal with such characters are provided. Some routines deal with a group of characters while others deal with individual characters, but most of them meet at common points and have common paths through the program. The routines and their actions are described below.

Reference /3/

At this entry a character is stored in the output word.

Reference /4/ - Exclamation Mark

Any spaces occuring between the ! and = are erased and the 'not equals' character (≠) output.

Reference /5/ - Underlined words

This routine tests a marker to determine whether the quotes (") is at the beginning or end of a word. If at the beginning, the next character is examined. If this next character is /, an integer divide character (÷) is output, if required.

Reference /6/ - Case Shift Characters

This routine deals with characters which occur in case shift in Flexowriter code. If the underlining marker is non zero, the character is preceded by an underline, assuming case shift. If the character is in a special string it is output regardless of the state of the case marker.

Otherwise the case marker is examined and a Case Shift output, if required, before the character.

Reference /7/ - Case Normal Characters

This is similar to reference /6/ in that the state of the case marker is checked and a Case Normal output if necessary before the character.

References /8/ and /9/ - Open and Close String quotes

The special string marker is set to zero if required and a Case Shift output if necessary before the string quote.

Reference /10/ - Special Strings

If the special string marker is non zero it is cleared and a Case Shift output. Otherwise it is set = -1 and a Case Normal output. The marker may be set to zero by the occurrence of string quotes or an end message.

Reference /12/ - Inequality

Having detected a 'less than' or 'greater than', the next character is examined. If it is 'equals' the appropriate inequality is output, preceded by an underline, otherwise the inequality is output. In both cases, a Case Shift is output if necessary.

Reference /11/ - End Message

The end message character is stored in the output word, which is then left justified and stored in the output buffer. The buffer is then written to magnetic tape. If this is the first end message (i.e. if EMMA = 1) control returns to reference /2/. If EMMA = 2, the marker DATA is set non zero to indicate that the size if the output buffer should be

reduced to 30 words and EWSA reset to 1. This allows the user to load several data tapes each terminated by an end message.

Reference /13/ - CRLF

This routine deals with the ambiguity arising from the fact that both 'Return' and 'minus' have the same internal six bit pattern. The next character in the input word is tested to see if it is a 'Line Feed'. If not, a 'minus' is output via reference /7/. If it is a Line Feed, any other consecutive occurences of this character are ignored and a CRLF character output via reference /3/.

Reference /80/ - Space in strings

The special string marker is examined and if it is zero, case shift is assumed and the 'space in strings (*) output. If the marker is not zero a Case Shift is output before the character and a Case Normal after it.

Reference /70/ - New Program Marker

The characters && indicate that a new program ~~has been~~ is being read in. All the existing markers and counters (except those referring to the input word and input buffer) are reset to zero. A single word separator block is written to magnetic tape together with a last block marker. If more than two consecutive ampersands are found another one word separator block and last block marker are written to the tape and the termination steps entered.

2.4.4 Input/output and preservation of Q stores

When Director is interrupted for any reason other than LOV, Q stores

5, 6 and 7 are preserved and therefore are available for use by Director. 07, however, is used to address the subprogram parameter table and therefore cannot be used by subprograms unless its contents are preserved by the subprogram. 06 is used by subroutine /RAU/, the device number of the allocated unit being left in 06. Due to these conditions, subprogram D0 uses 05 for all input and output transfers. 03 and 04 are also used by the subprogram, their contents being preserved on entry and restored prior to any peripheral operation since the machine may revert to program mode while the transfer is in progress. Before an input transfer is initiated, 06 is preserved and before an output transfer, 05 is preserved.

Information is read into a 32 word buffer, and the output from the program is written to magnetic tape in 32 word blocks if it is program and 30 word blocks if it is data. A single word of all zeros is written after each 30 word data block. The subroutines concerned with input and output are described below.

Subroutine /CHAR/

This routine takes a Flexowriter character in 01 and stores it in the output word. It also checks the state of the input and output buffers and the current input and output words.

Subroutine /NEXT/

This fetches the next word from the input buffer.

Subroutine /PERI/

When either or both of the input and output buffers have been processed, this subroutine is entered to organize the appropriate read or write.

Subroutine /WRIT/

This writes a buffer to magnetic tape. The data separator blocks are also written if required.

Subroutine /READ/

This reads 32 words from the paper tape reader into the input buffer.

Subroutine /SAD/

This routine preserves and restores the contents of Q3 and Q4.

2.4.5 Termination steps

A final separator block and last block marker are written to the magnetic tape which then backskips to Beginning of Tape Label. Backskip is used in preference to 'rewind' since the latter does not cause an EOT interrupt and does not check parity. The magnetic tape and the paper tape reader are then deallocated using the Director subroutine /RMDS/ and control passes to /RESP/ where the subprogram is closed down.

2.4.6 Constants and buffer areas

The following is a list of the constants and buffer areas and the purposes for which they are used in the program.

Constants

ENNA = count of the number of end messages read.

MEBO = -1 if End of Tape Warning on magnetic tape has been sensed or if a program separator block has just been written to the tape. Otherwise MEBO = 0.

DATA = 0 if a program is being processed and = -1 if data is being

processed. This indicates whether 32 or 30 word buffers should be used for output.

NOOP - One word program operator consisting of seven non-printing characters and an end message (in Flexowriter code).

IN - Input device number.

OUT - Output device number.

LD0,LD1 - Preserve subroutine return addresses (LINKS).

DN1,DN2 - Preserve cells N1, N2 of the nesting store.

DQ3,DQ4 - Preserve contents of Q3, Q4.

DQX - Preserve Q5 or Q6 during a peripheral transfer.

Q3 and Q4 are used by the subprogram and while the subprogram is operating, these Q stores contain the following information:

Q3:

C3 = number of characters in the input word, initially 6.

I3 = 0 if the last character output was in case shift, -1 if the last character output was in case normal.

N3 = -1 if the current character is to be underlined, otherwise 0.

Q4:

C4 = number of characters in the output word, initially 0.

I4 = -1 if the current character is in a special string, otherwise 0.

M4 = used to preserve a character when more than three cells of the next would be required.

Buffer areas

OBUF - general 32 word output buffer.

IBUF - 32 word input buffer.

BUF1 - 32 word output buffer for programs.

BUF2 - 30 word output buffer for data.

BUF3 - 1 word buffer for data separator and program separator.

There are only two 32 word buffer areas; BUF1, BUF2 and BUF3 are all subsets of OBUF.

2.5 Operational Aspects

In this section we consider some problems encountered in running the system and their effects.

2.5.1 Requirements

In order to run the batch processing system the Non-Time-Sharing Director with extended subprogram facilities must be present in the machine. For input and output, a magnetic tape (type 60 device) and a paper tape reader (type 67) are required. Programs and data are read in, converted to KDF9 tape code and stored on the magnetic tape. Programs thus loaded are then compiled and run by the modified Whetstone Compiler.

2.5.2 Running the system

On each afternoon that a practical class was held the students prepared and punched Algol programs which were then run in batches. A

computer operator supervised the data preparations and loaded the paper tapes.

On average four batches were run each day, three in the afternoon and one in the early evening. Each batch consisted on average of 25 programs and associated data, although the actual number varied considerably between 10 and 30. A section of the monitor typewriter output for a typical batch is shown in Appendix 3.

Many of the students used the Teletype equipment in the morning, when available, and thus had their tapes ready for the afternoon runs. Some batches, therefore, contained many of the previous day's programs, and results from batches run in the evening were available for the students the following morning.

2.5.3 Modifying the system

During the first term of 1966-67, a few minor alterations were made to the subprogram.

Each program run in the batch system must begin with two ampersands. It had been intended that no character, not even spaces, should occur between those ampersands. If spaces were present, the subprogram did not detect two consecutive ampersands and consequently the program separator block was not written on the magnetic tape.

When a program has been compiled and run, the Rhetotone Compiler will look for the separator block indicating the start of a new program. Thus if there is no separator block between two programs, the second

program will be ignored.

Several students typed spaces between the ampersands and naturally complained when their programs were not run. The routine dealing with ampersands (2.4.3, Reference /70/) was modified to allow for such spaces.

Another problem encountered was the rewinding of the magnetic tape and shut-down of the subprogram in the middle of a batch. The coding was scrutinised and various checks performed but no error in the subprogram was found. Eventually it was discovered that some students were winding up their tapes backwards! When such a program was loaded, after a program read in correctly, the subprogram found four consecutive ampersands and consequently terminated the run.

One major flaw in the representation of Algol and its implementation was discovered. On many occasions, students forgot to type quotes at the end of an 'underlined word', e.g. "BEGIN" was typed as "BEGIN. Words are underlined according to whether or not a marker word is zero or non-zero. Omitting the closing " leaves the marker set non-zero which causes all the following characters to be underlined, until the next " is found, when the new symbol will not be underlined as it should. The underlining on-off switch is thus 'on' when it should be 'off' and vice-versa. A typing error of this kind can give rise to several pages of the Algol Translator failure 'illegal underlined word'.

This problem arises because of the representation of underlined words, which is not context-free, and from its implementation. It would obviously be unfair to students to alter the representation of Algol in

the middle of a course and the best approach is to amend the batch loading program. The correction, which switches the underlining marker 'off' whenever a ~~space~~ CRLF ~~or card-skip~~ is output, has been incorporated into a Usercode version of DS for use on the Time-shared KDF9. The Time-shared batch processing system is considered in Chapter 3.

CHAPTER 3

THE TIME-SHARED SYSTEM

3.1 The Time-Shared KDF9

3.1.1 Upgrading of the KDF9

In the period January - March 1967, the KDF9 computer was upgraded to a larger configuration as recommended by the Flowers Committee. The main alterations were

(i) The addition of four extra modules of core store.

(ii) The addition of four more magnetic tapes, making nine in all.

(iii) Other peripheral equipment - a second line printer and a card punch.

(iv) The addition of 5.9 million words of disc storage.

(v) The provision of four levels of time-sharing by extra hardware.

With time-sharing facilities, up to four programs may be stored in core at one time, control passing between programs whenever any one program is held up (e.g. because of busy/lockout conditions) according to priority numbers assigned to programs.

The higher priority levels are usually occupied by programs with large amounts of input and/or output. These will be interrupted frequently, allowing lower priority programs to use the central processor. When a high priority program is no longer held up, a 'Program Ready' (PR) interrupt causes the current program to be suspended and the high priority program to be resumed.

3.1.2 Time-Sharing Director

A larger and more complex Director is necessary to control the running of program on the Time-Shared KDF9. Many of Director's functions such as interchanging priority levels and passing control between programs are organised by subprograms. The subprogram system, although similar in design to that in the Non-Time-Sharing Director, is consequently more complicated.

3.1.3 Batch processing

The loading of Teletype programs to magnetic tape on the upgraded KDF9 is performed by a Usercode program running in a high priority level. Because of the complexity of the subprograms in Director, pseudo-off-line loading is difficult to implement and, in any case, with four levels of time-sharing, is not necessary. Results are stored on magnetic tape and subsequently printed on the English Electric printer. This means that spurious 'X's do not appear and there is no longer any need for % in WRITE TEXT statements.

3.2 The Batch Loading Program

3.2.1 General alterations

Several alterations have been made to the batch loading program. It has been re-written in Usercode to operate in program mode. There is therefore no necessity for the program to preserve Q stores and cells of the nests. Code conversion is achieved by a much shorter 'look-at' table.

3.2.2 Input

The type number 67 has been restricted by English Electric to unlabelled magnetic tapes. The loading program now uses a paper tape reader (type 2).

3.2.3 Algol Representation

The use of % for special strings is now restricted to program and magnetic tape identifiers. This means that 'space in strings' (%) does not have to be looked for as a special character, since % may not be used within WRITE TEXT statements. This restriction simplifies matters both for the user and for code conversion purposes.

3.2.4 Code Conversion

In the subprogram B6, the 'look-at' table occupied 64 words of core store. The table has been reduced to 16 words, with some extra associated coding.

Each word in the table consists of four 12-bit groups. The first 6 bits in a group contain a positive integer used as an 'indicator' to the routine dealing with the character whose Flexowriter equivalent (if there is one), is stored in the second 6 bits of the group.

The indicators in octal and the corresponding characters in groups of characters are shown on the following page.

00 - space, semi-colon.

01 - exclamation mark.

02 - quotes (").

03 - case shift characters.

04 - case normal characters.

05 - string quotes.

06 - special strings (%).

07 - inequalities.

10 - Return, minus.

11 - End Message.

12 - ampersand.

The table is accessed by the value of the Teletype character, using the Usercode 'half-length fetch' instruction. The value of the character is halved. The quotient is used as a modifier to fetch the appropriate half word from the table and the remainder to select the correct 12 bits from this half word. The integer indicator is then placed in SJNBl and control passed (using EXIT) to a table of 11 jump instructions each of which leads to a routine to deal with the character groups in the above table.

3.2.5 Special Character Routines

Some changes have been made in these routines. The coding to deal with the % symbol has been removed as a result of the restriction mentioned in 3.2.3. Open and close string quotes are now dealt with in the routine

'string quotes' (reference 8 in the program) instead of separately as in D8. The markers for underlined words (M5) and special strings (I4) are now set to zero by the occurrence of a newline. Thus errors appearing on one line do not affect any other lines.

3.3 Results and conclusions

3.3.1 System efficiency

To compare the batch processing system with the standard Whetstone system, a batch of eleven programs were loaded onto magnetic tape and run in batch mode. Paper tape Flexowriter copies of these programs were obtained from the magnetic tape and run in the standard Whetstone system. The following table gives some indication of the differences in the systems.

	TOTAL RUN TIME	TOTAL ELAPSED TIME *	AVERAGE RUN TIME PER PROGRAM	AVERAGE ELAPSED TIME PER PROGRAM	RATIO OF RUN TO ELAPSED TIMES
BATCH SYSTEM	1m46s	3m31s	9.6s	19 secs	53%
STANDARD WHETSTONE	1m56s	6m56s	10.5s	47 secs	22%

* i.e. the time taken to translate and execute the batch of programs. During this time the computer is not available for any other job.

The first thing to be noticed is the decrease in the total elapsed time. Programs are processed much more quickly in batch mode than in the standard mode. For example a more typical batch of 33 programs would be run in 10½ minutes as against 27 minutes by the ordinary Whetstone system.

Better use of the central processor resulted, since the loading time

for these short programs was reduced by loading them from magnetic tape. In batch mode, a ratio of run time to elapsed time of 53% was achieved by the sample batch as against 22% in standard mode. This ratio, however, can still be improved. Much of the elapsed time in batch mode is accounted for by bringing into core the Translator and Controller segments of the compiler. With a large core store, both could be kept in core and control transferred between the segments. However, this is not usually possible, particularly on a multi-programmed machine. Time may be saved by having the compiler on the disc file, but owing to installation difficulties, the disc-based system has not so far been attempted.

The figures in the above table are for a sample batch of 11 programs. Two failed to compile, four compiled but failed at run time and five ran successfully. The ratio of run to elapsed time will naturally vary depending on the programs in the batch, but the figures indicate the value of the batch system.

3.3.2 Undergraduate teaching

The implementation of the batch processing system has created a method for processing quickly and easily a large number of short jobs produced by undergraduate students. A decrease in run time means that students receive their results more quickly. This in turn means that students get more runs. On some occasions, however, the turn-round time was not as good as was expected. Two factors contributed to this. Results were often stored on magnetic tape for subsequent printing, which did not always take place immediately. If a large tape was being printed while

the batch was being processed, the batch results would have to wait until the printing finished.

In addition, a lot of trouble arose with the Teletype equipment. Parity failures were a frequent cause for concern and some machines produced spurious characters. With only eight Teletypes available for 25 students each afternoon, the loss of even one machine was serious.

In spite of the difficulties, a large number of student programs were run. The largest batch processed contained 52 programs. Such a number would have been impossible to run on the standard Whetstone system.

3.4 The GOLD Project

3.4.1 Introduction

Experience has shown that Teletypes are not robust enough for normal off-line data preparation. They would, however, be suitable for on-line working and will be used as consoles in a multi-access system known as the GOLD system (GOLD : Glasgow On-Line Desks).

Using the multi-programming facilities of the upgraded KDF9 and a program assembly scheme called PROMPT, users will be able to establish both program and data files on disc. Bulk output will be sent to magnetic tape for subsequent printing while small and medium output will be sent back to the desks.

The on-line Teletypes will also be used by undergraduate students during practical classes in addition to the batch processing system. The existing batch system will in fact form a standby-system.

3.4.2 Code Conversion

Since Teletypes are being used to communicate with the KDF9, it is necessary to convert from Teletype code to KDF9 code and vice versa. The system programs comprising PROMPT deal with Algol Basic Symbols. Input is on paper tape in KDF9 code. Eventually, the 8 channel conversion tables will be replaced by Teletype tables, but in the meantime, incoming Teletype characters will be converted to KDF9 code before being handed over to PROMPT. In order to send results back to a desk, a conversion from KDF9 code to Teletype code has to be carried out. To accomplish code conversion, two subroutines, P300 and P301, have been written.

Subroutine P300 converts from Teletype code to Flexowriter code and is an extension of the code conversion section of the batch loading program. The incoming Teletype characters consist of 8 bits. The parity channel is removed (without checking) and the remaining 7 bits used to access the 32-word conversion table. The format of the table is the same as that described in 3.2.4.

Since 7 bits are used, there is no confusion between 'return' and 'minus' or 'multiply' and 'Line Feed' and the routine 'CRLF' (see 2.4.3) has been removed.

Subroutine P301 converts from Flexowriter code to Teletype code. The structure of the routine is similar to that of P301, i.e. a look-at table and a series of special character routines. Each 12-bit entry in a word of the table is made up of a 4-bit indicator and an 8-bit character.

3.5. Applications

One of the problems which has arisen both in the batch processing system and in the GOLD project has been the conversion from Teletype

code to KDF9 code and vice versa. This has been achieved by a look-at table and a series of routines to deal with special characters. Although the code conversion packages were written in Usercode, they could easily be written in Algol. The table could be contained in an array while the special character routines could be written as a set of procedures, these routines being entered by means of a switch.

It should therefore be possible for a user who finds himself with data preparation equipment which is not compatible with the computer he intends to use, to write his own conversion package.

The next stage in multi-access is the multi-processor system, i.e. a single computer with more than one processor or by linking several machines into a computer network. With such a system there comes the problem of communicating information in different codes between different machines. Somewhere between processor and user and also between processors, a code conversion must take place. Code conversion packages may be used in such systems to provide an interface until standardization comes along.

APPENDIX 1

PAPER TAPE CODES

KDF9 CODE

DECIMAL VALUE	FUNCTION
0	SPACE
1	
2	CRLF
3	PAGE THROW
4	TAB
5	
6	CASE SHIFT
7	CASE NORMAL
8	
9	
10	
11	
12	
13	
14	

DECIMAL VALUE	SYMBOL NORMAL/SHIFT	
15	/	:
16	0	↑
17	1	[
18	2	]
19	3	<
20	4	>
21	5	=
22	6	×
23	7	÷
24	8	(
25	9	)
26		
27	$\overline{10}$	£
28	;	;
29	+	≠
30	-	*
31	.	,

DECIMAL VALUE	SYMBOL NORMAL/SHIFT	
32		
33	A	a
34	B	b
35	C	c
36	D	d
37	E	e
38	F	f
39	G	g
40	H	h
41	I	i
42	J	j
43	K	k
44	L	l
45	M	m
46	N	n
47	O	o
48	P	p
49	Q	q
50	R	r
51	S	s
52	T	t
53	U	u
54	V	v
55	W	w
56	X	x
57	Y	y
58	Z	z
59	END FILE	
60	END DATA	
61	→	→
62		
63	DELETE	

TELETYPE CODE

DECIMAL VALUE	SYMBOL	DECIMAL VALUE	SYMBOL
32	SPACE	64	•
33	!	65	A
34	"	66	B
35	½	67	C
36	$	68	D
37	%	69	E
38	&	70	F
39	'	71	G
40	(	72	H
41	)	73	I
42	*	74	J
43	+	75	K
44	,	76	L
45	-	77	M
46	.	78	N
47	/	79	O
48	Ø	80	P
49	1	81	Q
50	2	82	R
51	3	83	S
52	4	84	T
53	5	85	U
54	6	86	V
55	7	87	W
56	8	88	X
57	9	89	Y
58	:	90	Z
59	;	91	[
60	<	92	£
61	=	93	]
62	>	94	↑
63	?	95	←

OTHER SYMBOLS USED ARE "RETURN"(DECIMAL VALUE 13) AND

"LINE FEED"(DECIMAL VALUE 10).

APPENDIX 2

A SPECIMEN ALGOL PROGRAM

FLEXOWRITER REPRESENTATION

```
DC01S009SPEC→
begin comment a program to produce the median of a set
                of n numbers. if n is even, the median is
                taken as the average of the middle two numbers;
      real median;
      integer n,i,q,r;
      open(20); open(70);
in:   n:=read(20); if n ≤ 0 then goto out;
      begin real array a[1:n];
         for i:= 1 step 1 until n do
         a[i]:=read(20);
         q:=n÷2;   r:=n-q×2;
         if r≠0 then median:= a[q]
         else median:= (a[q]+ a[q+1])/2;
         write text(70,[[cc] MEDIAN ** IS]);
         write (70,format([ndd.dddcc]), median);
      end;
      goto in;
out:  close(20); close(70);
end→
```

TELETYPE REPRESENTATION

```
&&
%DC01S009SPEC←

"BEGIN COMMENT" A PROGRAM TO PRODUCE THE MEDIAN OF A SET OF
                N NUMBERS. IF N IS EVEN, THE MEDIAN IS TAKEN
                AS THE AVERAGE OF THE MIDDLE TWO NUMBERS;

      "REAL" MEDIAN;

      "INTEGER" N,I,Q,R;

      OPEN(20); OPEN(70);

IN:   N:= READ(20); "IF" N < = 0 "THEN GOTO" OUT;

      "BEGIN REAL ARRAY" A[1:N];

          "FOR" I:= 1 "STEP" 1 "UNTIL" N "DO"

          A[I]:= READ(20);

          Q:= N "/" 2;    R:= N-Q*2;

          "IF" R !=0 "THEN" MEDIAN:=A[Q]

          "ELSE" MEDIAN:= (A[Q]+A[Q+1])/2;

          WRITE TEXT(70,\\CC/ %MEDIAN $$ IS /);

          WRITE (70,FORMAT(\NDD.DDDCC/), MEDIAN);

      "END";

      "GOTO" IN;

OUT:  CLOSE(20); CLOSE(70);

"END"←
```

APPENDIX 3

MONITOR OUTPUT FOR A TYPICAL BATCH

N10 15/04 1644
N10 M KMW020111UPU
N10 DC01U007C401→
N10 DC01U007R980→
N10 RAN/EL/000M09S/000M10S
N10 DC01U007G8A2→
N10 DC01U007R69A→
N10 RAN/EL/000M13S/000M15S
N10 DC01U007R68A→
N10 DC01U007K1PD→
N10 DC01U007C490→
N10 DC01U007M7F2→
N10 DC01U007R664→
N10 RAN/EL/000M12S/000M21S
N10 DC01U007E29A→
N10 DC01U007B961→
N10 DC01U007C685→
N10 RAN/EL/000M02S/000M05S
N10 DC01U007M4A1→
N10 DC01U007E5LX→
N10 DC01U007C480→
N10 RAN/EL/000M08S/000M09S
KMW020111UPU ENDS 0
N10RAN/EL/001M23S/003M30S

APPENDIX 4

BIBLIOGRAPHY

APPENDIX A

Bibliography

BROOKER, R.A., and ROHL, J.S. (1966).

"Experience with a first-year computer science course".

Computer Bulletin, Vol.10, No. 3, p.42.

GLOOT, P.L. (1965).

"What is the use of operating systems?"

Computer Journal, Vol.7, No. 4, p.249.

FISHER, F.P., and SWINDLE, G.F. (1964).

"Computer Programming Systems"

Holt, Rinehart, Winston, London and New York.

KILBURN, T., et. al. (1961).

"The Manchester University Atlas Operating System, Part 1: Internal Organisation".

Computer Journal, Vol.4, No. 3, p.222.

LYNCH, W.C. (1966).

"Description of a High Capacity, Fast Turnaround University Computing Centre."

Comm. ACM, Vol.9, No. 2, p.117.

PAMELL, B.K. (1966).

"Operating systems - a critical view."

Office Management, Winter 1966.

RANDELL, B., and RUSSELL, L.J. (1964).

"Implementation of Algol 60"

Academic Press, London and New York.

WEGNER, P. (Ed) (1964).

"Introduction to System Programming" Chapter 7.

"The Whetstone KDF9 Algol Translator" (B. RANDELL)

Academic Press, London and New York.

KDF9 Documentation

(1) Non-Time-Sharing Director Support Documentation.

(2) KDF9 Programming Manual.

(3) KDF9 Algol Programming.

(4) Service Routine Library Manual, Vol.2, section 10.

APPENDIX 5

PROGRAMS

DIRECTOR SUBPROGRAM D8

```
(TELETYPE LOADING SUBPROGRAM);

R0; /A000/; BLANK; SYL3,B23;
/A001/; SYL3;

(INITIAL SETTING UP);
SET B67; JS/RAU/; J/RESP/; C6; =E/IN/;        (CLAIM READER);
SET B60; JS/RAU/; J/78/; C6; =E/OUT/;         (CLAIM MAG. TAPE);

/1/;  (CLEAR MARKERS AND COUNTERS);
      ZERO; DUP; DUP;=E/DATA/; =E/MTBC/; =E/EMMA/;
      JS/READ/; JS/QSD/; Q0 TO Q3; Q0 TO Q4;
      E/BUF1/; DUP; =E/OBUF/; SHL-16; =RM6;
      SET32; DUP; =C5; =C6; SET8; DUP; =C3; =C4;
/83/; ZERO; MOM5Q;
/2/;  (ENTRY TO TABLE);
      ZERO; SHLD6; DC3; =LINK; EXIT/100/;
/3/;  (OUTPUT A CHARACTER);
      JS/CHAR/; J/99/; J/2/;
/99/; JS/PERI/; J/198/; J/2/;
```

```
*/100/;  (CODE CONVERSION LOOK-AT TABLE);

        SET0; J/3/;          J/4/;          *J/5/;
             *J/2/;*SETB36;J/80/; SETB7; J/10/;
             J/70/;*SETB22; J/9/; SETB30; J/6/;
      SETB31; J/6/; SETB26; J/6/; SETB35; J/7/;
      SETB37; J/6/;        J/13/;*SETB37; J/7/;
      SETB17; J/7/; SETB20; J/7/; SETB21; J/7/;
      SETB22; J/7/; SETB23; J/7/; SETB24; J/7/;
      SETB25; J/7/; SETB26; J/7/; SETB27; J/7/;
      SETB30; J/7/; SETB31; J/7/; SETB17; J/6/;
      SETB34; J/3/; SETB23;J/12/; SETB25; J/6/;
      SETB24;J/12/; SETB33; J/7/; SETB21; J/8/;
      SETB41; J/6/; SETB42; J/6/; SETB43; J/6/;
      SETB44; J/6/; SETB45; J/6/; SETB46; J/6/;
      SETB47; J/6/; SETB50; J/6/; SETB51; J/6/;
      SETB52; J/6/; SETB53; J/6/; SETB54; J/6/;
      SETB55; J/6/; SETB56; J/6/; SETB57; J/6/;
      SETB60; J/6/; SETB61; J/6/; SETB62; J/6/;
      SETB63; J/6/; SETB64; J/6/; SETB65; J/6/;
      SETB66; J/6/; SETB67; J/6/; SETB70; J/6/;
      SETB71; J/6/; SETB72; J/6/; SETB21; J/6/;
      SETB33; J/6/; SETB22; J/6/; SETB20; J/6/;
      SETB75; J/11/;
```

```
(SPECIAL CHARACTER ROUTINES);

/4/; (EXCLAMATION MARK);
/14/;   J/15/C3Z; ZERO; SHLD6; DC3; J/14/=Z; SET B35; J/6/;
/15/;   JS/NEXT/; J/14/C5NZ; JS/PERI/; J/198/; J/14/;

/5/; (UNDERLINED WORDS);
        M3; NOT; =M3;
/93/;   J/16/C3Z;
/17/;   M3; J/2/=Z;
/66/;   ZERO; SHLD6; REV; =E/DN1/; ZERO; J/90/=; SET B17;
        J/67/NE; ERASE; E/DN1/; DC3; SET B27; =M4; J/68/;
/67/;   E/DN1/; REV;  SHLD-6; ERASE; I3; J/2/=Z;
        IO TO Q3; SETB6; J/3/;
/16/;   JS/NEXT/; J/17/C5NZ; JS/PERI/; J/198/; J/17/;
/90/;   DC3; ERASE; E/DN1/; J/93/;

/6/; (CASE SHIFT CHARS.);
        =M4; M3; J/18/NEZ; I4; J/19/NEZ;
/68/;    I3; J/19/=Z; IO TO Q3; SETB6;
/20/;    JS/CHAR/; J/98/;
/19/;    M4; J/3/;
/18/;    SET B32; J/20/;
/98/;    JS/PERI/; J/198/; J/19/;

/7/; (CASE NORMAL CHARS.);
      =M4; I3; J/19/NEZ; I3=-1; SET B7; J/20/;
```

```
/8/; (OPEN STRING QUOTES);
      =M4;
/25/; I3; J/18/=Z;IO TO Q3; IO TO Q4;
      SET B6; JS/CHAR/; J/97/; J/18/;
/97/; JS/PERI/; J/198/; J/18/;

/9/;  (CLOSE STRING QUOTES);
      =M4; I4; J/25/=Z; IO TO Q4; J/25/;

/10/; (SPECIAL STRING MARKER);
      =M4; I4; J/58/NEZ;I4=-1; I3=-1; J/19/;
/58/; IO TO Q3; IO TO Q4; SET B6; J/3/;

/12/; (INEQUALITY);
       =M4; I3; J/27/=Z; IO TO Q3;
       SET B6; JS/CHAR/; J/95/; J/28/;
/45/;  ERASE; E/DN1/; DC3;
/27/;  J/29/C3Z;
/28/;  ZERO; SHLD6; REV; =E/DN1/; ZERO; J/45/=;
       SETB35; J/30/=; E/DN1/;REV; SHLD-6; ERASE; J/19/;
/29/;  JS/NEXT/; J/28/C5NZ;
/95/;  JS/PERI/; J/198/; J/28/;
/30/;  ERASE; DC3; E/DN1/; J/18/;
```

```
/13/;  (CRLF);
       J/31/C3Z;
/32/;  ZERO; SHLD6; REV; =E/DN1/;
       SETB12;  J/33/NE; DC3; ERASE; E/DN1/; J/34/C3Z;
/35/;  ZERO; SHLD6; REV; =E/DN1/;
       SETB12; J/36/NE; DC3; ERASE; E/DN1/; J/35/C3NZ;
/34/;  JS/NEXT/; J/35/C5NZ; JS/PERI/; J/198/; J/35/;
/31/;  JS/NEXT/; J/32/C5NZ; JS/PERI/; J/198/; J/32/;
/36/;  E/DN1/; REV; SHLD-6; ERASE; SETB2; J/3/; (CRLF);
/33/;  E/DN1/; REV; SHLD-6; ERASE; SETB36; J/7/;(MINUS);

/80/; (SPACE IN STRINGS);
      =M4;I3; J/19/=Z; SET 6; JS/CHAR/; J/94/;
/81/; =E/DN1/; M4; SET 7; =M4; E/DN1/; REV; J/20/;
/94/; JS/PERI/; J/198/; J/81/;

/11/; (END MESSAGE);
      JS/CHAR/; J/38/;
/38/; =E/DN1/; IO TO Q4;
/55/; SHL6; DC4; J/55/C4NZ; =MOM6;
/39/; JS/QSD/; Q5; =E/DQS/; JS/WRIT/; J/198/; E/DQS/; =Q5;
/40/; E/EMMA/; NOT; NEG; SET 2; J/43/NE;
/41/; NEG; NOT; DUP; NEG; =E/DATA/; E/BUF2/; DUP;
      =E/OBUF/; SHL-16; =RM6; SET 30; =C6; (DATA MODE);
/43/; =E/EMMA/; JS/QSD/; SET 8; =C4; ZERO; E/DN1/; J/2/;
```

```
/79/; ERASE; E/DN1/;

/70/;   (NEW PROGRAM MARKER);
        J/77/C3Z;
/71/;   ZERO; SHLD6; DC3; REV; =E/DN1/; ZERO; J/79/=;
        SET 6; J/72/=; =LINK; E/MTBC/; J/73/NEZ;
/74/;   E/DN1/; EXIT/100/;

/73/;   (RESET MARKERS AND START NEXT PROGRAM);
        IMO TO Q3; ZERO; DUP;=E/DATA/;
        DUP; =E/EMMA/; =E/MTBC/; IO TO Q4;
/76/;   E/BUF1/; DUP; =E/OBUF/; SHL-16; =RM6;
        SET 32; =C6; SET 8; =C4; J/74/;

/72/;   (WRITE BLOCK BETWEEN PROGRAMS);
        ERASE; ERASE; Q5; =E/DQS/; JS/QSD/;
/198/; (ENTER HERE IF ETW);
        E/BUF3/; =Q5; E/MC05/; =MOM5; E/OUT/; =C5;
        POCQ5; JS/RWL5/; E/MTBC/; J/75/NEZ; ZERO; NOT;
        =E/MTBC/; JS/QSD/; E/DQS/; =Q5; ZERO; E/DN1/; J/70/;
/77/;   JS/NEXT/; J/71/C5NZ; JS/PERI/; J/198/; J/71/;
```

```
(INPUT/OUTPUT ROUTINES AND Q-STORE PRESERVATION);

/CHAR/; (STORE CHAR.);
        SHC-6; CAB; SHLD6; DC4; PERM; ERASE;
        J/Z1/C4Z; J/Z2/C3Z; EXIT 2;
/Z1/; REV; =MOM6Q; SET8; =C4;
        ZERO; REV; J/Z3/C3NZ;
/Z2/; ERASE; MOM5Q; SET8; =C3;
/Z3/; J/Z4/C6Z; J/Z4/C5Z; EXIT 2;
/Z4/; EXIT 1; (BUFFER PROCESSED);

/QSD/; (DUMP AND RETRIEVE Q3,Q4);
        E/DQ3/; Q3; =E/DQ3/; =Q3;
        E/DQ4/; Q4; =E/DQ4/; =Q4; EXIT 1;

/NEXT/; (INPUT WORD EMPTY);
        ERASE; MOM5Q; SET8; =C3; EXIT 1;

/READ/; (INPUT BUFFER EMPTY);
         LINK; =E/LD1/; E/IBUF/; =Q5; E/IN/; =C5;
         JS/RWL5/; PIAQ5; JS/RWL5/;
         I5; =RM5; SET 32; =C5;
/Z48/;   E/LD1/; =LINK; EXIT 1;
```

```
/WRIT/; (OUTPUT BUFFER FULL);
        LINK; =E/LD1/; E/OBUF/; =Q5; E/OUT/; =C5;
        POBQ5; JS/RWL5/;PMFQ5; J/Z61/TR; E/DATA/; J/Z59/NEZ;
        E/OBUF/; SHL-16; =RM6; SET32; =C6; J/Z60/;
/Z59/;  (DATA SEPARATOR);
        E/BUF3/; =Q5; ZERO; =MOM5; E/OUT/; =C5;
        POBQ5; JS/RWL5/;PMFQ5; J/Z61/TR;
        E/OBUF/; SHL-16; =RM6; SET30; =C6;
/Z60/;  E/LD1/; =LINK; EXIT 2;
/Z61/;  ZERO; NOT; =E/MTBC/;E/LD1/; =LINK; EXIT 1;


/PERI/; (BUFFER PROCESSED);
       =E/DN1/; =E/DN2/; LINK; =E/LDO/;
        JS/QSD/; J/P1/C6NZ; Q5; =E/DQS/;
        JS/WRIT/; J/P6/; E/DQS/; =Q5; J/P2/C5NZ;
/P1/;   Q6; =E/DQS/; JS/READ/; E/DQS/; =Q6;
/P2/;   JS/QSD/; E/DN2/; E/DN1/; E/LDO/;=LINK; EXIT 2;
/P6/;   E/LDO/; =LINK; EXIT 1;  (ETW);


(TERMINATION STEPS);

/75/; POCQ5; JS/RWL5/; QO TO Q5; E/OUT/; =C5;
      PMEQ5; JS/RWL5/; E/OUT/; JS/RMDE/; JS/RTP/;
/78/;  E/IN/; JS/RMDE/; JS/RTP/; J/RESP/;
```

```
(BUFFERS AND CONSTANTS);

*/EMMA/;   SYL6;      (END MESSAGE COUNT);
/MTBC/;    SYL6;      (ETW/LAST BLOCK);
/LDO/;     SYL6;      (LINK DUMP);
/LD1/;     SYL6;      (LINK DUMP);
/DN1/;     SYL6;      (DUMP N1);
/DN2/;     SYL6;      (DUMP N2);
/DATA/;    SYL6;      (PROGRAM/DATA MARKER);
/DQS/;     SYL6;      (Q-STORE DUMP);
/IN/;      SYL6;      (READER);
/OUT/;     SYL6;      (MAG. TAPE);
/DQ3/;     SYL6;      (DUMP Q3);
/DQ4/;     SYL6;      (DUMP Q4);
/MC05/;    SYL6, B 0505050505050575;  (PROGRAM SEPARATOR);

/IBUF/; SYL2; SYL2,/AOOB/; SYL2,/AOOB+31/;     (INPUT BUFFER);
/BUF1/; SYL2; SYL2,/AOOB+32/; SYL2,/AOOB+63/; (OUTPUT PROGS);
/BUF2/; SYL2; SYL2,/AOOB+32/; SYL2,/AOOB+61/; (OUTPUT DATA);
/BUF3/; SYL2; SYL2,/AOOB/; SYL2,/AOOB/;       (SEPARATOR);
/OBUF/; SYL6;      (GENERAL OUTPUT BUFFER);


RO; /AOOB/; SYL6;
RO; SYL6;
RO; /A111/;
BLANK;
(END OF D8);
```

THE BATCH LOADING PROGRAM

```
P
LOADTELOOUP2
LOAD TELETYPE PROGS. TO MAG. TAPE IN FLEX. CODE→
ST1000;
TL7200;
V25;
YZ63;
PROGRAM;

(buffers and constants);

V0 = Q0/AYZ0/AYZ31;          (input buffer);
V1 = Q0/AYZ32/AYZ63;         (output prog. buffer);
V2 = Q0/AYZ32/AYZ61;         (output data buffer);
V3 = Q0/AYZ32/AYZ32;         (indicator word);
(V4 = prog/data marker,
 V5 = end message count,
 V6 = etw/last block present,
 V7 = general output buffer);
V8 = B 0505050505050575;      (program separator);
V9 = Q0/0/AV10;               (base address of table);
```

```
(code conversion table);

V10 = B 00 00 01 77  02 77 00 77;
V11 = B 03 36 06 07  12 77 05 22;
V12 = B 03 30 03 31  03 26 04 35;
V13 = B 03 37 10 36  04 37 04 17;
V14 = B 04 20 04 21  04 22 04 23;
V15 = B 04 24 04 25  04 26 04 27;
V16 = B 04 30 04 31  03 17 00 34;
V17 = B 07 23 03 25  07 24 04 33;
V18 = B 05 21 03 41  03 42 03 43;
V19 = B 03 44 03 45  03 46 03 47;
V20 = B 03 50 03 51  03 52 03 53;
V21 = B 03 54 03 55  03 56 03 57;
V22 = B 03 60 03 61  03 62 03 63;
V23 = B 03 64 03 65  03 66 03 67;
V24 = B 03 70 03 71  03 72 03 21;
V25 = B 03 33 03 22  03 20 11 75;

(initial setting up);

101;    SET B60; SET 5; OUT; DUP; =RC3; M+I3;
        PMAQ3; (skip label block);
        SHL32; DUP; V1; OR; =V1;
102;    DUP; V2; OR; =V2; V3; OR; =V3;
        SET 2; SET 5; OUT; SHL32; V0; OR; =V0;
```

```
(set up markers,counters and output buffer);

  1;  ZERO; DUP; =V4; DUP; =V5; =V6; JS155;   (initial read);
      QO TO Q3; QO TO Q4; V1; DUP; =V7;
      SHL-16; =RM6; SET 32; =C6; SET 8; DUP; =C3; =C4;
 83;  ZERO; MOM5Q;
  2;  (fetch character from table);
      ZERO; SHLD6; DC3;
 84;  SHC-1; DUP; SHL+1; SHA-1; V9; =M4;
      M4; SHL+1; +; =M4; MOM4H;
 21;  REV; J23≥Z; SHL+12;
 23;  ZERO; SHLD6; REV; SHL-42; REV;
      SHC-1; DUP; =LINK; J24≥Z; EXIT 1 AR100;
 24;  EXIT AR100;

*100;     J3; J4; J5; J6; J7; J8;
          J10; J12; J13; J11; J70;

  3;  (output a character);
          JS150; J2;
(special character routines);
  4;      (exclamation mark);
 14;      ERASE;
 22;      J15C3Z; ZERO; SHLD6; DC3; J22 = Z; SET B35; J6;
 15;      JS152; J22;
```

```
 5;      (underlined words);
         ERASE;M3; NOT; = M3;
93;      J16C3Z;
17;      M3; J2 = Z;
66;      ZERO; SHLD6; ZERO; J90 =;
         SET B17; J67 ≠; ERASE; DC3; SET B27; = M4; J68;
67;      SHLD - 6; ERASE; I3; J2 = Z; IO TO Q3; SET B6; J3;
90;      DC3; ERASE; J66C3NZ;
16;      JS152; J17;

 6;      (case shift chars.);
         =M4; M3; J18 ≠ Z; I4; J19 ≠ Z;
68;      I3; J19 = Z; IO TO Q3; SET B6;
20;      JS150;
19;      M4; J3;
18;      SET B32; J20;

 7;      (case normal chars.);
         =M4; I3; J41 ≠ Z; I3 = -1; SET B7; J20;
41;      M3; J18≠Z; J19;

 8;      (string quotes);
         =M4;
25;      I3; J18 = Z; IO TO Q3; SET B6; JS150; J18;

10;      (special string marker);
         =M4;I4 = -1; I3 = -1; J19;
```

```
12;     (inequality);
63;     = M4; I3; J27 = Z; IO TO Q3; SET B6; JS150; J28;
62;     ERASE; DC3;
27;     J29C3Z;
28;     ZERO; SHLD6; ZERO; J62 =;
40;     SET B35; J30=; SHLD-6; ERASE; J19;
29;     JS152; J28;
30;     ERASE; DC3; J18;

13;     (crlf);
        ERASE; J31C3Z;
32;     ZERO; SHLD6; SET B12; J33 ≠; DC3; ERASE; J34C3Z;
35;     ZERO; SHLD6; SET B12; J36 ≠; DC3; ERASE; J35C3NZ;
34;     JS152; J35;
31;     JS152; J32;
36;     SHLD - 6; ERASE; SET B02; IOTOQ4; MOTOQ3; J3; (crlf);
33;     SHLD - 6; ERASE; SET B36; J7; (minus);

11;     (end message);
        JS150; IO TO Q4; REV;
55;     SHL6; DC4; J55C4NZ; = MOM6; ZERO; REV;
44;     JSP1; JS157; V5; NOT; NEG; SET 2; J43≠;
46;     NEG; NOT; DUP; NEG; =V4;          (set data marker);
42;     V2; DUP; =V7; SHL-16; =RM6; SET 30; =C6;
43;     =V5; SET 8; =C4; J2;
```

```
70;     (new program marker);
        ERASE;
77;     J71C3NZ; JS152;
71;     ZERO; SHLD6; DC3; ZERO; J70=;
        SET 6; J72=; V6; J84=Z;

73;     (reset markers for new program);
74;     IMO TO Q3; ZERO; DUP; DUP; =V4; = V5; = V6;
76;     IO TO Q4; V1; DUP; =V7; SHL-16; =RM6; SET 32; =C6;
        SET 8; =C4; J84;

72;     (write block between programs);
        ERASE; REV; ERASE; V3; =Q6; V8; =MOM6;
78;     POCQ6; V6; J75≠Z; ZERO; NOT; =V6; ZERO; REV; J77;

(termination steps);

75;     ERASE; POCQ6; POCQ6; IMO TO Q6; PMEQ6;
        VO; SHL-32; SET 6; OUT;
        V1; SHL-32; SET 6; OUT;
        ZERO; OUT;
```

```
150;     (Store chars.and test buffers);
         SHC - 6; CAB; SHLD6; DC4; PERM; ERASE;
         J151C4Z; J152C3Z; EXIT 1;
151;     (output word full);
         REV; = MOM6Q; SET 8; = C4; ZERO; REV; J153C3NZ;
152;     (input word empty);
         ERASE; MOM5Q; SET 8; = C3;
153;     J154C6Z;  J155C5Z; EXIT 1;
154;     (output buffer full);
         JSP1; J157; J156C5NZ;
155;     (input buffer empty);
         VO; =Q5; PIAQ5; PARQ5; J155TR; I5; =RM5; SET 32; =C5;
156;      EXIT 1;
157;     (etw);
        LINK; ZERO; NOT; =V6; J72;


P1;     (write to mag. tape);
        V7PO; DUP; =Q6; POBQ6; PMFQ6; J61TR;
        V4PO; J59≠Z; SHL-16; =RM6; SET 32; =C6; J60;
 59;    V3PO; =Q6; ZERO; =MOM6; POBQ6; PMFQ6; J61TR;
        SHL-16; =RM6; SET 30; =C6;
 60;    EXIT 2;
 61;    ERASE; EXIT 1;(etw);


FINISH;→
```